

Ah, PIVOT and UNPIVOT in Oracle SQL – powerful tools for reshaping your data! Let's dive into them.

PIVOT

Imagine you have data where values from one column need to become new column headers. That's where `PIVOT` comes in. It essentially rotates rows into columns.

Syntax:

SQL

```
SELECT *
FROM (
    -- Subquery or table containing the data
    SELECT pivot_column, grouping_column, aggregate_column
    FROM your_table
)
PIVOT (
    aggregate_function(aggregate_column)
    FOR pivot_column
    IN (value1 AS new_column1, value2 AS new_column2, ...)
);
```

Explanation of Syntax:

- `SELECT * FROM (...)` : You start with a subquery or directly select from a table. This result set needs to contain the columns you want to pivot.
- `PIVOT (...)` : This is the core of the `PIVOT` operation.
- `aggregate_function(aggregate_column)` : This specifies the aggregate function (like `SUM`, `AVG`, `COUNT`, `MAX`, `MIN`) to be applied to the `aggregate_column`. For each unique value in the `pivot_column`, a new column will be created, and this function will determine the value in that new column based on the corresponding `grouping_column`.
- `FOR pivot_column IN (value1 AS new_column1, value2 AS new_column2, ...)` :
 - `pivot_column` : This is the column whose distinct values will become the new column headers.
 - `IN (value1 AS new_column1, value2 AS new_column2, ...)` : This explicitly lists the values from the `pivot_column` that you want to transform into new columns. You can optionally

provide aliases (AS new_column1) for these new columns to make them more readable. If you omit the AS clause, the new column names will be the values themselves.

Example:

Let's say you have a table called sales_data :

Product	Quarter	Sales
A	Q1	100
B	Q1	150
A	Q2	120
B	Q2	180
A	Q3	90
B	Q3	200

Export to Sheets

Here's how you can use PIVOT to see the sales of each product by quarter:

SQL

```
SELECT *
FROM (
  SELECT product, quarter, sales
  FROM sales_data
)
PIVOT (
  SUM(sales)
  FOR quarter
  IN ('Q1' AS Q1_Sales, 'Q2' AS Q2_Sales, 'Q3' AS Q3_Sales)
);
```

Output:

PRODUCT	Q1_SALES	Q2_SALES	Q3_SALES
A	100	120	90
B	150	180	200

Export to Sheets

Where to use PIVOT?

- **Generating cross-tab reports:** When you need to present data in a tabular format where one column's values become column headers.
- **Data analysis:** To compare metrics across different categories (e.g., sales by region, product performance over time).
- **Simplifying complex queries:** Sometimes, pivoting can make data easier to understand and work with in subsequent queries or applications.

Why use PIVOT?

- **Improved readability:** Pivoted data can be more intuitive to read and interpret, especially for end-users.
- **Simplified reporting:** It can streamline the process of creating reports that require this specific data layout.
- **Easier comparisons:** Comparing values across different categories becomes straightforward when they are presented in separate columns.

UNPIVOT

`UNPIVOT` performs the reverse operation of `PIVOT`. It transforms multiple columns into rows. If you have a table where certain attributes are represented as columns, `UNPIVOT` can convert them back into rows, making the data more normalized.

Syntax:

SQL

```
SELECT *  
FROM your_table  
UNPIVOT (  
    value_column FOR attribute_column  
    IN (column1 AS attribute1, column2 AS attribute2, ...)  
);
```

Explanation of Syntax:

- `SELECT * FROM your_table UNPIVOT (...)` : You select from the table you want to unpivot.

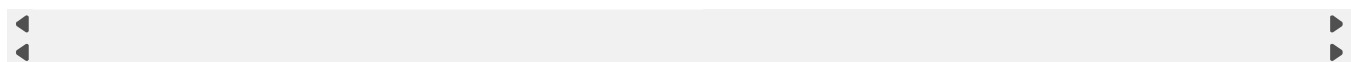
- `UNPIVOT (...)` : This is the core of the `UNPIVOT` operation.
- `value_column FOR attribute_column` :
 - `value_column` : This is the name you want to give to the new column that will contain the values from the columns you are unpivoting.
 - `attribute_column` : This is the name you want to give to the new column that will contain the original column names (the "attributes").
- `IN (column1 AS attribute1, column2 AS attribute2, ...)` :
 - `column1, column2, ...` : These are the names of the columns you want to unpivot.
 - `AS attribute1, AS attribute2, ...` (optional): You can provide aliases for the original column names, which will appear in the `attribute_column` . If you omit the `AS` clause, the original column names will be used.

Example:

Let's take the output of the previous `PIVOT` example as our starting point (let's call this table `quarterly_sales`):

PRODUCT	Q1_SALES	Q2_SALES	Q3_SALES
A	100	120	90
B	150	180	200

Export to Sheets



Here's how you can use `UNPIVOT` to transform it back to a structure similar to the original `sales_data` table:

SQL

```
SELECT *
FROM quarterly_sales
UNPIVOT (
    Sales FOR Quarter IN (Q1_Sales AS 'Q1', Q2_Sales AS 'Q2', Q3_Sales AS 'Q3')
);
```

Output:

PRODUCT	SALES	QUARTER
A	100	Q1
A	120	Q2
A	90	Q3
B	150	Q1
B	180	Q2
B	200	Q3

Export to Sheets

Where to use UNPIVOT?

- **Normalizing denormalized data:** When you receive data in a wide format with attributes as columns and you need to transform it into a more relational (long) format.
- **Preparing data for applications or analysis:** Some applications or analytical tools might expect data in a specific row-based format.
- **Simplifying complex queries:** If you need to perform operations across what were originally columns (e.g., finding the maximum value across different periods), unpivoting can make this easier.

Why use UNPIVOT?

- **Data normalization:** It helps in converting data into a more normalized structure, reducing redundancy and improving data integrity.
- **Flexibility in querying:** Once data is unpivoted, you can easily query and analyze it based on the original attributes (now in a single column).
- **Compatibility with other tools:** It can make data more compatible with tools and systems that expect a row-oriented format.

In essence, `PIVOT` reshapes data from rows to columns for better reporting and comparison, while `UNPIVOT` does the opposite, transforming columns back into rows for normalization and more flexible querying. They are valuable tools in your SQL arsenal for manipulating and preparing data in Oracle.

You got it! Here are some real-world scenarios where `PIVOT` and `UNPIVOT` in Oracle SQL prove incredibly useful:

Real-World Scenarios for PIVOT

1. Sales Reporting by Region and Product Category:

Imagine a company that sells various products across different geographical regions. Their sales data might be stored in a table like this:

Sale ID	Region	Category	Sales Amount
1	North	Electronics	1200
2	South	Clothing	800
3	North	Clothing	950
4	East	Electronics	1500
5	West	Home Goods	600
6	South	Electronics	1100

Using `PIVOT`, you can transform this data to generate a report showing the total sales amount for each product category in each region:

SQL

```
SELECT *
FROM (
    SELECT region, category, "Sales Amount"
    FROM sales_data
)
PIVOT (
    SUM("Sales Amount")
    FOR region
    IN ('North' AS North_Sales, 'South' AS South_Sales, 'East' AS East_Sales, 'West' AS West_Sales)
);
```

Output:

CATEGORY	NORTH_SALES	SOUTH_SALES	EAST_SALES	WEST_SALES
Clothing	950	800		
Electronics	1200	1100	1500	
Home Goods			600	

This pivoted view makes it easy to compare the performance of different product categories across various regions.

2. Website Traffic Analysis by Day and Traffic Source:

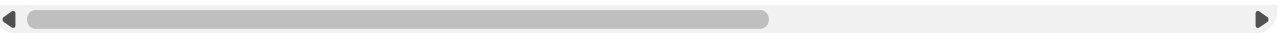
Consider a website that tracks its traffic from different sources (e.g., direct, organic search, paid ads, social media) on a daily basis. The raw data might look like:

Date	Source	Visits
2025-05-10	Direct	500
2025-05-10	Organic Search	800
2025-05-11	Direct	550
2025-05-11	Paid Ads	300
2025-05-11	Social Media	400
2025-05-12	Organic Search	900

Using `PIVOT`, you can generate a report showing the number of visits from each source for each day:

SQL

```
SELECT *
FROM (
    SELECT trunc(date) AS visit_date, source, visits
    FROM website_traffic
)
PIVOT (
    SUM(visits)
    FOR source
    IN ('Direct' AS Direct_Visits, 'Organic Search' AS Organic_Visits, 'Paid Ads' AS P
)
ORDER BY visit_date;
```



Output:

	VISIT_DATE	DIRECT_VISITS	ORGANIC_VISITS	PAID_ADS_VISITS	SOCIAL_MEDIA_VISITS
	:-----	:-----	:-----	:-----	:-----
	2025-05-10	500	800		
	2025-05-11	550		300	400
	2025-05-12		900		

This pivoted view helps in quickly understanding the performance of different traffic sources over time.

Real-World Scenarios for UNPIVOT

1. Analyzing Student Grades Across Different Subjects:

Consider a table storing student grades for various subjects:

Student ID	Math	Science	English	History
:-----	:--	:-----	:-----	:-----
101	85	92	78	88
102	76	80	91	79
103	90	88	82	95

If you need to find the average grade across all subjects for each student or identify the highest grade achieved by each student, it's easier to work with the data in a long format.

UNPIVOT can transform this:

SQL

```
SELECT student_id, subject, grade
FROM student_grades
UNPIVOT (
    grade FOR subject IN (Math, Science, English, History)
);
```

Output:

STUDENT_ID	SUBJECT	GRADE
:-----	:-----	:-----
101	Math	85
101	Science	92
101	English	78
101	History	88

102	Math	76	
102	Science	80	
102	English	91	
102	History	79	
103	Math	90	
103	Science	88	
103	English	82	
103	History	95	

Now, you can easily perform aggregate functions across all subjects for each student.

2. Processing Survey Data with Multiple Responses:

Imagine a survey where respondents can select multiple options for a particular question, and these responses are stored in separate columns:

Response ID	Favorite Color 1	Favorite Color 2	Favorite Color 3	
:-----	:-----	:-----	:-----	
1	Red	Blue		
2	Green	Yellow	Orange	
3	Blue			

To analyze the frequency of each color chosen, you would need to combine these columns into a single column. `UNPIVOT` helps here:

SQL

```
SELECT response_id, favorite_color
FROM survey_responses
UNPIVOT (
    favorite_color FOR color_number IN ("Favorite Color 1" AS Color1, "Favorite Color
)
WHERE favorite_color IS NOT NULL;
```

Output:

RESPONSE_ID	FAVORITE_COLOR	
:-----	:-----	
1	Red	
1	Blue	
2	Green	
2	Yellow	

2	Orange	
3	Blue	

This unpivoted data makes it easy to count the occurrences of each favorite color.

These examples highlight how `PIVOT` and `UNPIVOT` can simplify data analysis and reporting by reshaping data according to specific requirements in various real-world scenarios. They are powerful tools for transforming data into a more usable format for different analytical tasks.